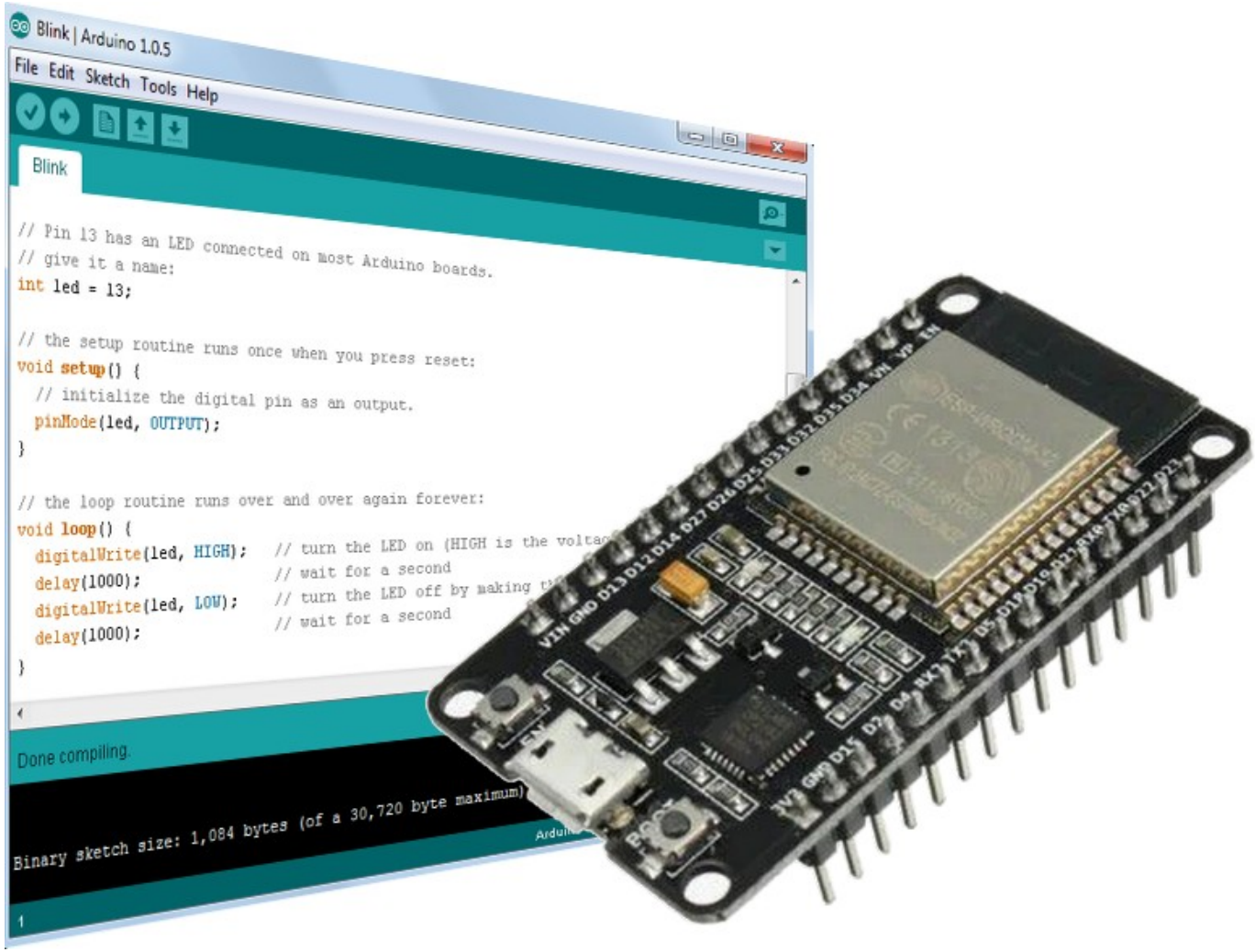
ESP32 mikrovezérlők programozása Arduino környezetben

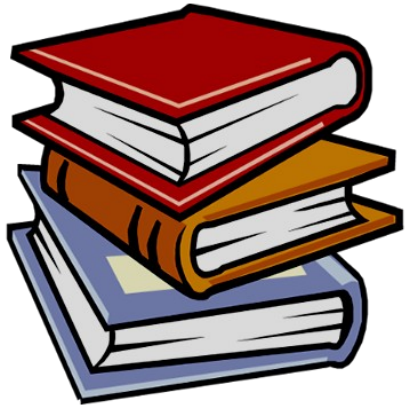


ARDUINO™

5. programmegszakítások, időzítők, digitális szenzorok

Felhasznált és ajánlott irodalom

- Harsányi Réka – Juhász Márton András:
  Fizikai számítástechnika – elektronikai alapok és Arduino programozás
- Last Minute Engineers:
  Configuring & Handling ESP32 GPIO Interrupts In Arduino IDE
- Rui Santos: ESP32 with PIR Motion Sensor using Interrupts and Timers
- DIYprojects: How to use Timers and Alarms with Arduino Code
- Last Minute Engineers: Interface Multiple DS18B20s with ESP32
- Espressif: ESP32 Technical Reference Manual



Példaprogramok

ESP32_button_interrupt–megszakítás nyomógombbal

ESP32_button_bounce–nyomógomb pergés vizsgálata

ESP32_PIR_sensor – infravörös mozgásérzékelő

ESP32_timer_alarm – periodikus megszakítás keltése

ESP32_button_debounce – periodikus mintavételezés

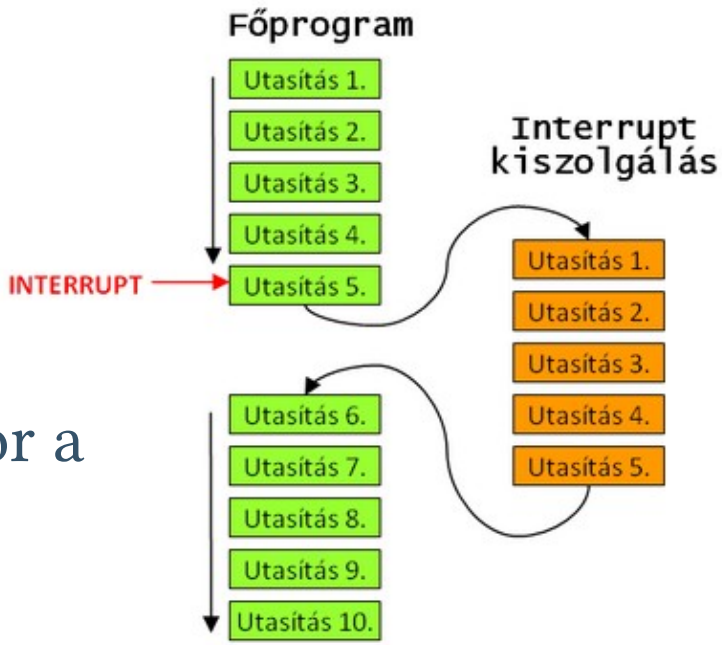
ESP32_BMP180 – légnyomás és hőmérséklet mérése

ESP32_BME280 – hőmérséklet,
légnyomás és páratartalom mérése

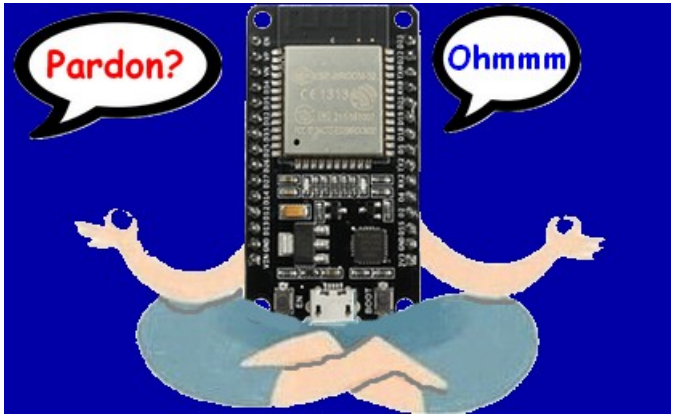
ESP32_DS18B20 – digitális hőmérő

Mi a programmegszakítás?

- **Programmegszakítás (interrupt):** egy hardver esemény hatására megszakad a program futása, egy előre elkészített függvény kiszolgálja az eseményt, majd folytatódik a félbehagyott program



- A programmegszakítás olyan, mint amikor a telefon csöng. Abbahagyjuk, amit éppen csináltunk, s felvesszük a kagylót.
  A telefonálás után ott folytatjuk a korábbi tevékenységet, ahol félbeszakítottuk, amikor a telefont felvettük
- A megszakított tevékenység lehet ”alvás” is (valamilyen szintű energiatakarékos mód)



Megszakítások kezelése Arduino programokban

- **Arduino Uno,** illetve **Arduino Nano:** csak két külső forrás használható: **INT 0 (D2)** és **INT 1 (D3)**
- **attachInterrupt**(*interrupt, ISR, mode*) – hozzárendel egy kiszolgáló eljárást (ISR) a megnevezett megszakításhoz és beállítja a megszakítás módját (`LOW, HIGH, FALLING, RISING, CHANGE`)
- **detachInterrupt**(*interrupt*) – megszünteti a korábbi hozzárendelést
- **digitalPinToInterrupt**(*pin*) – **Arduino** esetén ez konverziós függvény, ahol *pin* a felhasznált kivezetés sorszáma, a visszatérési érték pedig az interrupt sorszáma

- **ESP32:** mindegyik GPIO tüske lehet megszakításkérő bemenet
- **ESP32** esetén a **digitalPinToInterrupt()** függvénynek szükségtelen a használata és megszakítási sorszám helyett közvetlenül a GPIO kivezetés sorszámát kell megadni az **attachInterrupt**(), illetve **detachInterrupt**() függvényekben

ESP32_button_interrupt.ino

- A beépített LED (GPIO2) vezérlése a beépített nyomógombbal (GPIO0)
- A megszakítási esemény a gomb lenyomása (`FALLING`)
- A gyors kiszolgálás érdekében a RAM területen tároljuk a **blink()** függvényt (`IRAM_ATTR` módosító)
- Megoldatlan maradt a nyomógomb pergésének a kezelése

```
volatile byte led_state = LOW;          // Állapotváltozó

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);         // LED_BUILTIN esetünkben GPIO2
  //Megszakítás a GPIO0 bemenetre kötött nyomógomb lenyomásakor
  attachInterrupt(0, blink, FALLING); // A kiszolgáló függvény hozzárendelése
}

void loop() { }

void IRAM_ATTR blink() {                // Megszakítás kiszolgálása
  led_state = !led_state;               // állapotváltás minden megszakításkor
  digitalWrite(LED_BUILTIN, led_state);
}
```

ESP32_button_bounce.ino

- A GPIO5 bemenetre kötött nyomógomb pergésének vizsgálata

```
#define button 5                    // Külsö nyomógomb (GPIO 5 és GND közé kötve)
volatile int counts;                // Számláló

void IRAM_ATTR button_pressed() { //--- Megszakítások kiszolgálása
  counts++;                         // megszakítások számlálása
}

void setup() {
  Serial.begin(115200);
  pinMode(button, INPUT_PULLUP);
  Serial.println("Press and release the builtin pushbutton!\r\n");
}

void loop() {
  counts = 0;                       // Számláló nullázása
  attachInterrupt(button, button_pressed, FALLING); // Kiszolgálás hozzárendelése
  while (digitalRead(button));      // Lenyomásra várunk
  delay(20);                        // pergésmentesítő várakozás
  while (!digitalRead(button));     // Felengedésre várunk
  delay(20);                        // pergésmentesítő várakozás
  detachInterrupt(button);          // Hozzárendelés megszüntetése
  Serial.print("Button pressed "); // Az eredmény kiírása
  Serial.print(counts);             <-- Erre vagyunk kíváncsiak!
  Serial.println(" times\r\n");
}
```

Egy lenyomás-felengedés ciklust vizsgálunk

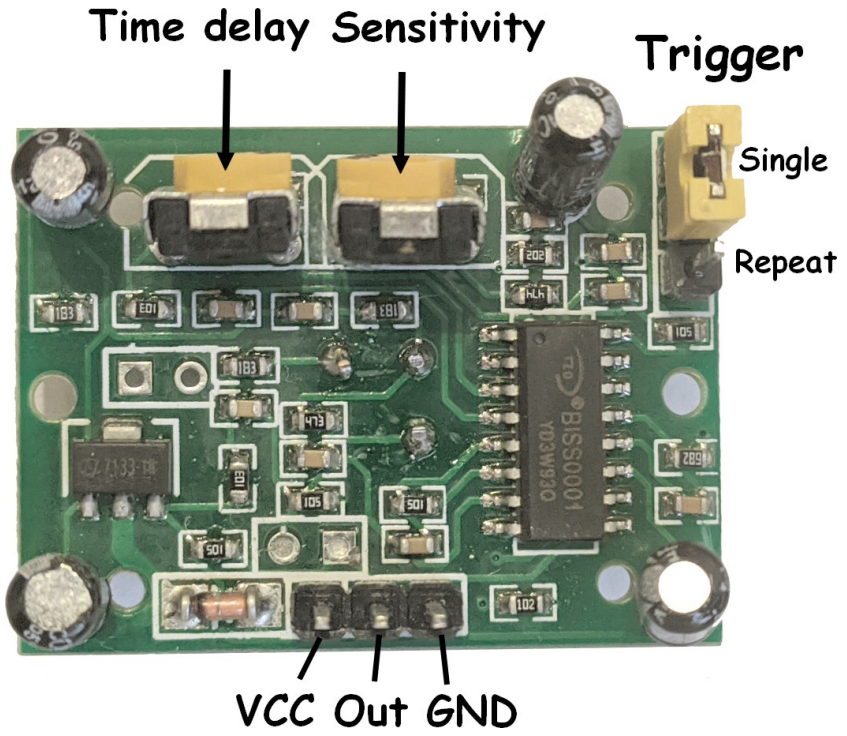
ESP32_button_bounce.ino

- **Tapasztalat:** a beépített nyomógomb esetében nem detektáltunk pergést, külső nyomógomb használatakor viszont durva különbségek mutatkoztak – szükséges tehát foglalkozni vele

```
Press and release the builtin pushbutton!
Button pressed 1 times
Button pressed 2 times
Button pressed 2 times
Button pressed 2 times
Button pressed 2 times
Button pressed 2 times
Button pressed 1 times
Button pressed 2 times
Button pressed 2 times
Button pressed 2 times
Button pressed 3 times
Button pressed 2 times
Button pressed 1 times
Button pressed 6 times
Button pressed 3 times
Button pressed 7 times
Button pressed 5 times
Button pressed 4 times
Button pressed 7 times
Button pressed 1 times
```

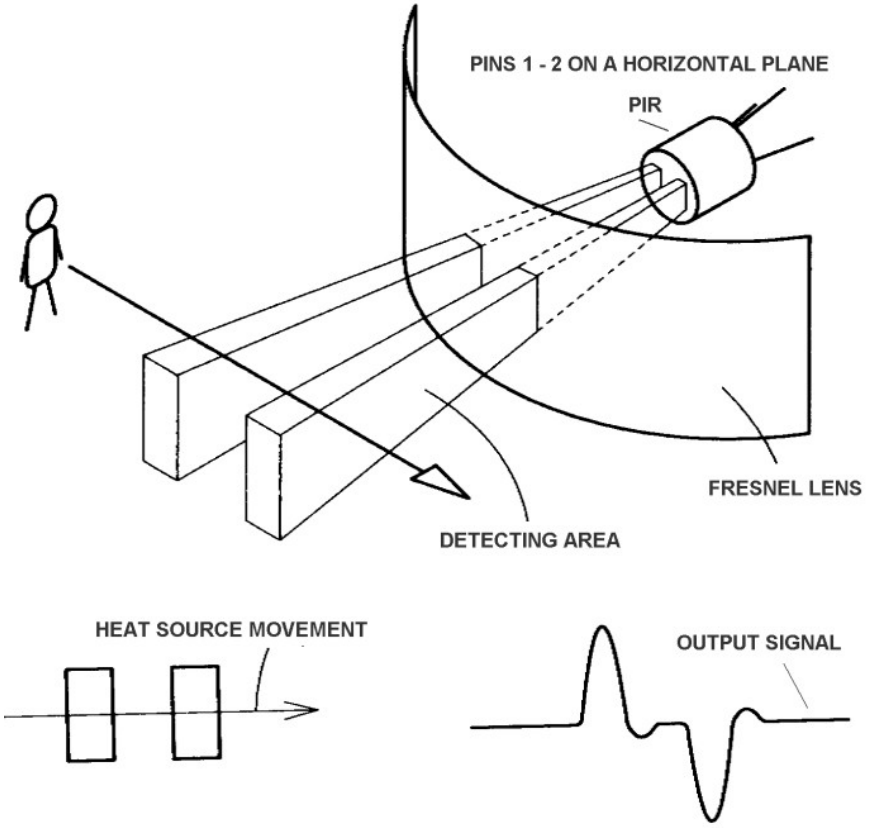
HC-SR501 PIR mozgásérzékelő

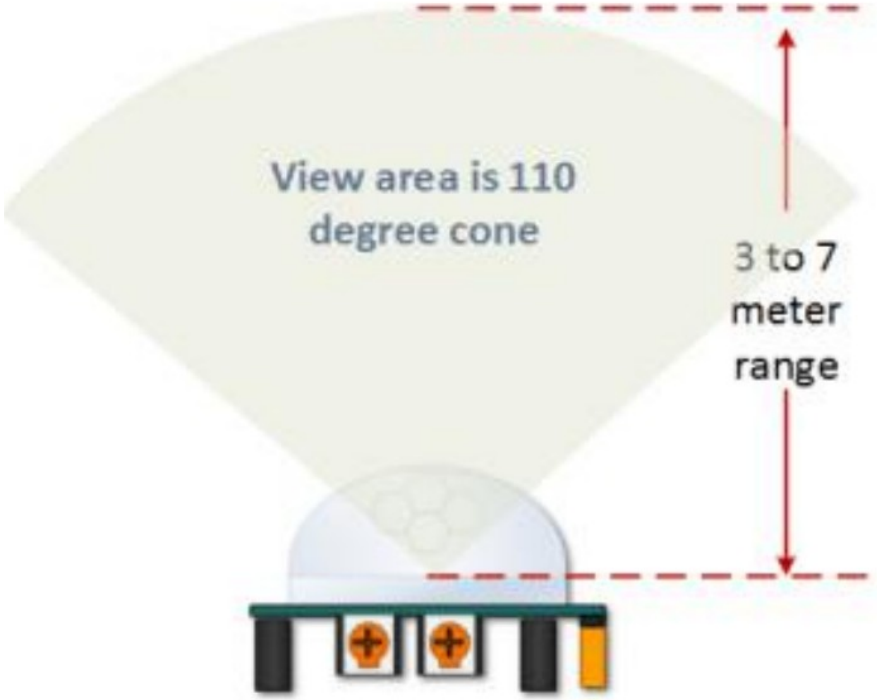
- A PIR (Passive Infrared) piroelektromos szenzor a testek infravörös hőmérsékleti sugárzását érzékeli.
- Tápfeszültség: 4,5 – 20 V
- Kimenő jel 3,3 V
- Érzékenység: 3 – 7 m
- Késleltetés: 3 s – 5 min
- Holtidő: 3 s
- Trigger mód:
  - egyszeri késleltetés
  - újrainduló késleltetés
- Komponensek:
  - D210S IR szenzor
  - BISS0001 IC



HC-SR501 PIR mozgásérzékelő

- A **D210S** detektor két cellájának differenciális jelét vizsgálva mozgást érzékelhetünk
- A látószöget egy Fresnel lencse tágítja ki 110°°-ra
- Az érzékenység és a késleltetési idő fokozatmentesen állítható





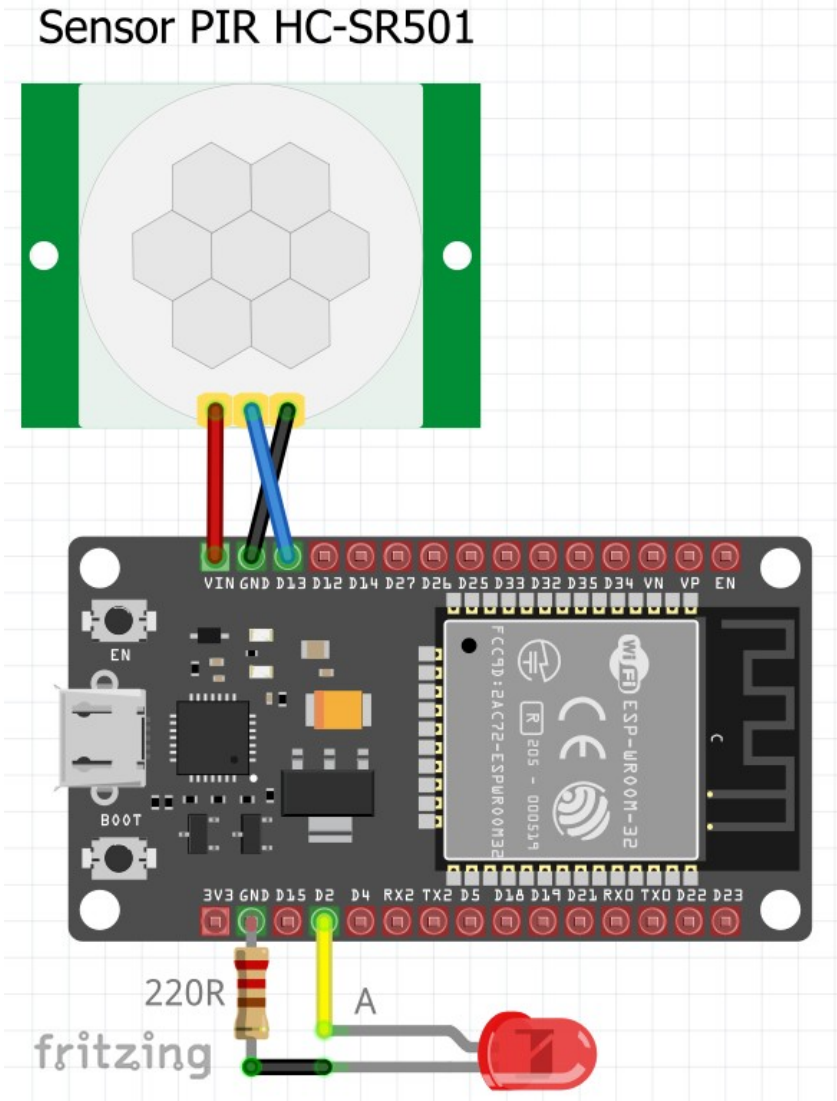
HC-SR501 PIR mozgásérzékelő

	Jobbra forgatva Nő a késleltetési idő, véghelyzetben kb. 5 perc
	Balra forgatva Csökken a késleltetési idő, véghelyzetben kb. 3 sec

	Jobbra forgatva Csökken az érzékenység, véghelyzetben kb. 3 m
	Balra forgatva Növekszik az érzékenység, véghelyzetben kb. 7 m

ESP32_PIR_sensor

- Ebben a projektben egy **HC-SR501** PIR szenzorral keltünk külső megszakításokat (felfutó élre kell érzékenyíteni a bemenetet)
- Megszakítás hatására a program előre megadott ideig végez valamilyen tevékenységet (itt a LED világít)
- Az időzítéshez a **millis()** függvényt használjuk
- A bekötési vázlat az ábrán látható
  Vin (5V) ⟶ VCC
  D13 ⟵ Vout
  GND ⟷ GND
  D2 ⟶ LED anód
  GND ⟵ LED katód
  (220R ellenálláson keresztül)



ESP32_PIR_sensor.ino

```
#define timeSeconds 5
const int led = 2;
const int motionSensor = 13;
unsigned long now, lastTrigger = 0;
volatile boolean startTimer = false;

void IRAM_ATTR detectsMovement() {
  digitalWrite(led, HIGH);       // Set alarm signal
  startTimer = true;             // Set timer flag
  lastTrigger = millis();        // Notice time of event
}

void setup() {
  pinMode(motionSensor, INPUT_PULLUP);
  attachInterrupt(motionSensor, detectsMovement, RISING);
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
}

void loop() {
  now = millis();
  if (startTimer && (now - lastTrigger > (timeSeconds * 1000))) {
    digitalWrite(led, LOW);
    startTimer = false;
  }
  delay(50);
}
```

Időzítők (Timers)

- Az ESP32 4 db általános célú időzítővel rendelkezik, amelyek 16 bites előszámlálóból és 64 bites számlálóból állnak
- Az előosztókkal a számláló órajele állítható elő a 80 MHz-es bemenőjelből történő leosztással (leosztás: 2 – 65 536 között)
- Arduino programokban az Alarm (riasztás) funkcióval kelthetünk megszakítást (amikor a számláló értéke megegyezik, vagy meghaladja az Alarm regiszterbe írt számot)
- **hw_timer*** – a hardver számláló objektumra mutató típus
- **timerBegin**(*timerID, divider, countUp*) – timer konstruktor
- **timerAttachInterrupt**(*timer, &ISR, edge*) – megszakítás hozzárendelés
- **timerAlarmWrite**(*timer, interruptAt, autoreload*) – Alarm beállítása
- **timerAlarmEnable**(*timer*) – riasztás engedélyezése

Ahol ***timerID*** = 0..3, ***timer*** = hw_timer* típusú mutató,
divider 16 bites előosztási szám, ***countUP, edge, autreload*** = boolean típusú,
interruptAt = 64 bites szám

ESP32_timer_alarm.ino

```
boolean led_state = false;                          // LED állapotjelző
volatile int count;                                 // Trigger
#define LED_PIN 2
hw_timer_t * timer = NULL;                          // Ez lesz a számláló fogantyúja

// Megszakítást kiszolgáló eljárás
void IRAM_ATTR onTime() {
  count++;
}

void setup() {
  pinMode(LED_PIN, OUTPUT);                         // LED kimenet konfigurálás
  digitalWrite(LED_PIN, LOW);
  timer = timerBegin(0, 80, true);                  // Timer0, 80x előosztás
  timerAttachInterrupt(timer, &onTime, true);
  timerAlarmWrite(timer, 1000000, true);            // Másodpercenként jelezzen
  timerAlarmEnable(timer);
}

void loop() {
  if (count > 0) {                                  // Ha idő van...
    count--;                                        // Jelző törlése
    led_state = !led_state;                         // LED állapot átváltása
    digitalWrite(LED_PIN, led_state);               // LED állapot aktualizálása
  }
}
```

- Periodikus megszakításokat keltünk. Felhasznált forrás: https://diyprojects.io/esp32-timers-alarms-interrupts-arduino-code/

- Ha jelzést kapunk a riasztásról (count > 0), átbillentjük a beépített LED állapotát

ESP32_button_debounce.ino

- „Javítsuk meg” az **ESP32_button_interrupt.ino** programot, azaz pergésmentesítsük a nyomógombot periodikus mintavételezéssel!
- **A pergésmentesítés elve** az, hogy csak bizonyos időnként (itt most 20 ms-onként) vizsgáljuk a bemenet állapotát, így az első lenyomás érzékelése és a következő mintavételezés közben már lezajlik a kontaktus fizikai pergése, tehát már stabilizálódott állapotot látunk
- A periodikus mintavételezések időzítéséhez **Timer0** megszakításait fogjuk használni
- **Az előosztás** 80 lesz, így a számláló 80/80 MHz időközönként, azaz 1 mikroszekundumonként lép egyet
- Az **Alarm** regiszterbe 20 000-et írunk, így 20 ms-onként keltünk megszakítást
- A lenyomás vagy felengedés érzékeléséhez a nyomógomb pillanatnyi állapotát az előző állapottal kell összehasonlítani

ESP32_button_debounce.ino

```
#define BUTTON      0                                   // Beépített nyomógomb (GPIO 0)
uint8_t button_state = 1;                               // Nyomógomb állapotjelző
volatile boolean keypress = false;                      // Gomblenyomás eseményjelző
boolean led_state = false;                              // LED állapot tárolója
hw_timer_t* timer = NULL;                               // Timer "fogantyú"
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;  // Mutex = kölcsönös kizárás

void IRAM_ATTR onTime() {                               // Megszakítás kiszolgálás
  button_state = (button_state << 1) | digitalRead(BUTTON);
  if ((!keypress) && ((button_state & 3) == 2))  // 1 -> 0 átmenet lenyomást jelez
  {
    portENTER_CRITICAL_ISR(&timerMux);                  // Kritikus szekcióba lépünk
    keypress = true;                                    // keypress írását védjük
    portEXIT_CRITICAL_ISR(&timerMux);                   // Kritikus szekció vége
  }
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);                         // LED kimenet konfigurálása
  digitalWrite(LED_BUILTIN, LOW);                       // és alaphelyzetbe állítása
  pinMode(BUTTON, INPUT_PULLUP);                        // Nyomógomb bemenet aktiválás
  timer = timerBegin(0, 80, true);                      // Timer0, előosztó=80
  timerAttachInterrupt(timer, &onTime, true);     // Kiszolgáló függvény csatolása
  timerAlarmWrite(timer, 20000, true);                  // Ébresztés 20 ms-onként
  timerAlarmEnable(timer);                              // Ébresztés engedélyezése
}
```

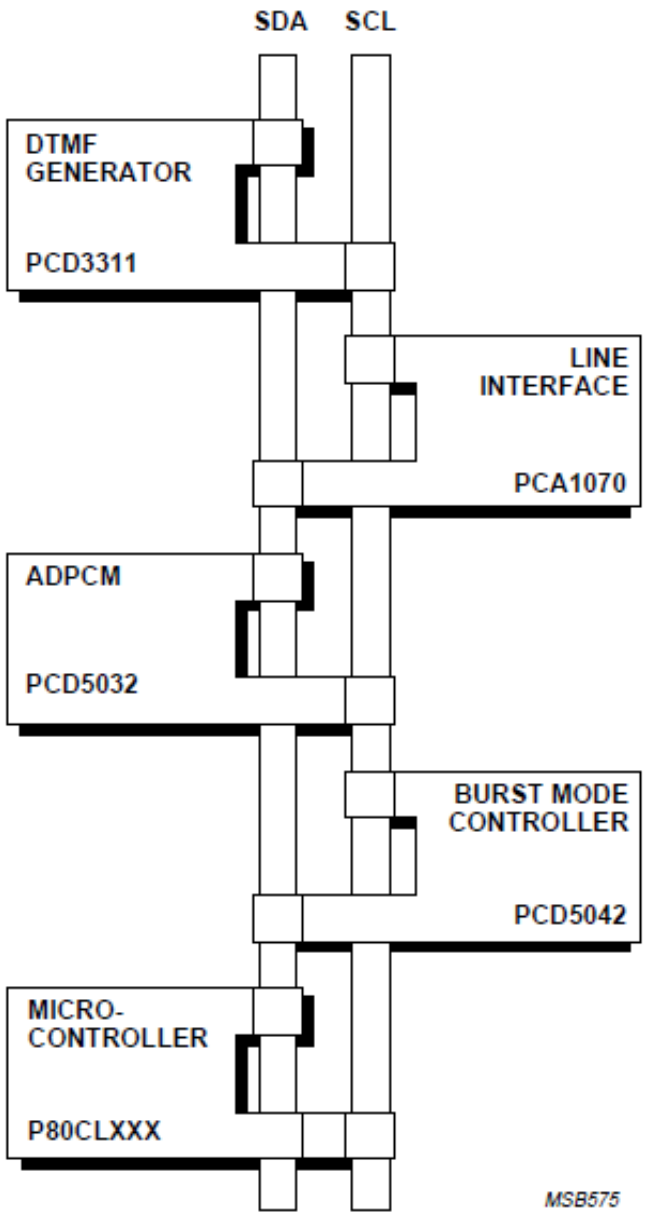
ESP32_button_debounce.ino

```
void loop() {
  if (keypress) {                                  // lenyomás esemény esetén
    portENTER_CRITICAL_ISR(&timerMux);             // kritikus szekció kezdete
    keypress = false;                              // jelző törlése
    portEXIT_CRITICAL_ISR(&timerMux);              // kritikus szekció vége
    led_state = !led_state;                        // LED állapot negálás
    digitalWrite(LED_BUILTIN, led_state);          // LED állapot aktualizálása
  }
}
```

- A programban egyúttal azt is bemutattuk, hogy hogyan védhetjük meg a változónkat módosítás közben a konkurens folyamatoktól
- A **mutex** (a *mutual exclusion* névből) egy szinkronizációs mechanizmus ami ideiglenesen korlátozza a konkurens folyamatokat egy adott erőforrás elérésében
- A programban a **keypress** változó tartalmát két helyen is módosítjuk, ezek „összeakadását” akadályozhatjuk meg egy mutex segítségével: a változót az írhatja, aki le tudta foglalni a mutex-et

Az I2C busz

- „Inter-Integrated Circuit” busz – vagy „kétvezetékes busz”, amelyet eredetileg a Philips cég dolgozott ki 1982-ben.
- **Több eszköz** (master és slave) **fűzhető fel a buszra**
- A buszt a **mester (master) eszközök** vezérlik és kezdeményezik az adatforgalmat. **A szolga (slave) eszköz** akkor válaszol, ha címzéssel megszólítják
- **Az I²C busz két jelvezetéket használ**
  - **SCL:** szinkronizáló órajel
  - **SDA:** soros adat
- **A részletes leírás** az ”Az I²C-busz specifikációja és használata” című dokumentumban olvasható



Hol találhatunk bővebb információt?

- Itt most csak érintőlegesen foglalkozunk az **I2C** busz használatával, részletesebb információ a korábbi előadásokban található

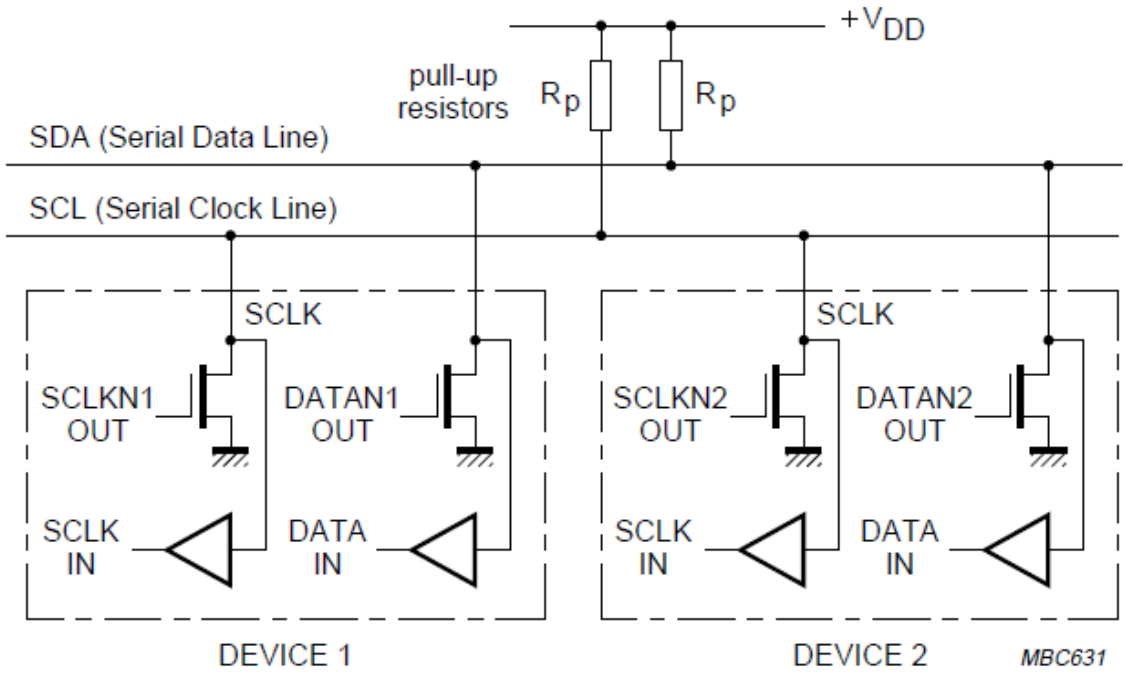
Arduino témakörű előadások:

- Az I2C kommunikációs csatorna (2020. február 6.)
  előadásvázlat mintaprogramok

STM32 mikrovezérlő témakörű előadások:

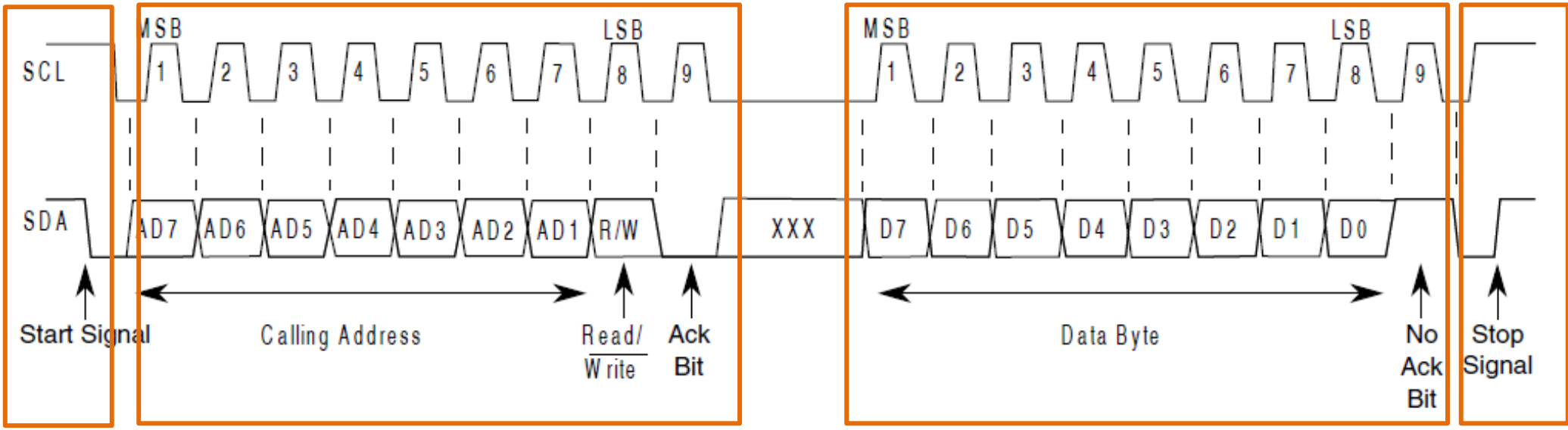
- Az I2C kommunikációs csatorna – 1. rész (2021. február 4.)
  előadásvázlat mintaprogramok video
- Az I2C kommunikációs csatorna – 2. rész (2021. február 18.)
  előadásvázlat mintaprogramok video

Az I2C busz vezérlése



- A jelvezetékeket ellenállások húzzák fel tápfeszültségre V_{DD}
- Nyitott nyelőelektródás FET-ek húzzák le a vonalakat alacsony szintre
- A buszt vezérlő mester eszköz állítja elő az SCL órajelet
  - normál mód: 100 kHz
  - gyors mód: 400 kHz
  - nagysebességű mód: 1 MHz, vagy több, ez esetben már aktív felhúzással.

I2C üzenetformátum



Üzenet-orientált adatátvitel négy felvonásban:

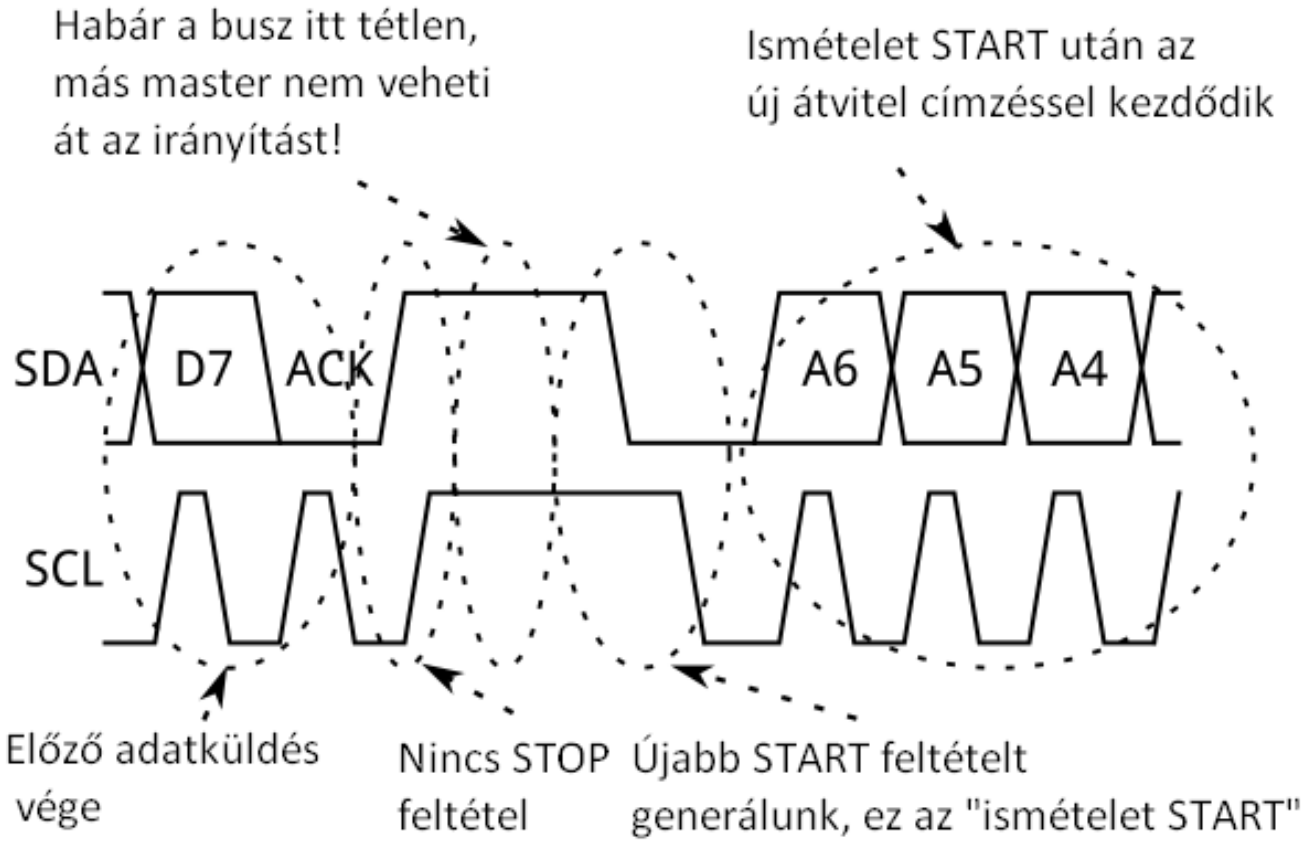
1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

Nyugtázás (ACK): a 9. órajel impulzus tartamára a vevő alacsony szinten tartja az **SDA** jelvezetéket.

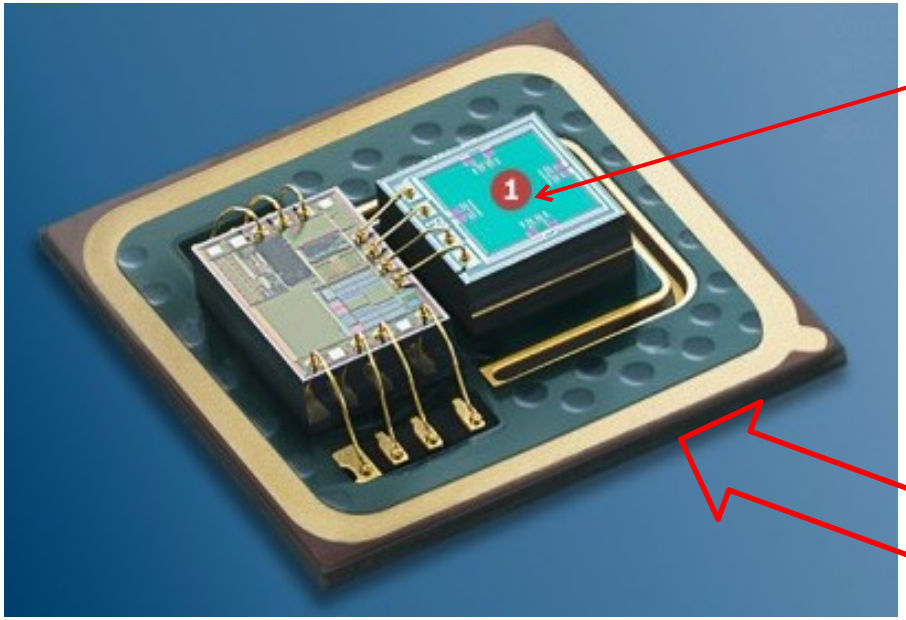
Negatív nyugtázás (NAK): a 9. órajel impulzus idején senki sem húzza le az **SDA** jelvezetéket – az magas szinten marad.

Ismételt START feltétel (repeated Start)

Gyakran szükség van arra, hogy egy összetett tranzakciót egy master egy menetben végigvihessen. Ilyen eset lehet például egy eszköz valamely regiszterének vagy memóriaterületének megcímzése, majd onnan adatok beolvasása. Ha az adatküldés és adatfogadás között nem generálunk **STOP** feltételt, akkor a master magánál tudja tartani a busz feletti vezérlést. A **STOP** nélkül generált újabb **START** feltételt **ismételt START**-nak (repeated START) nevezzük. Az **ismételt START** feltételt újabb cím/parancs bájt küldése kell, hogy kövesse.

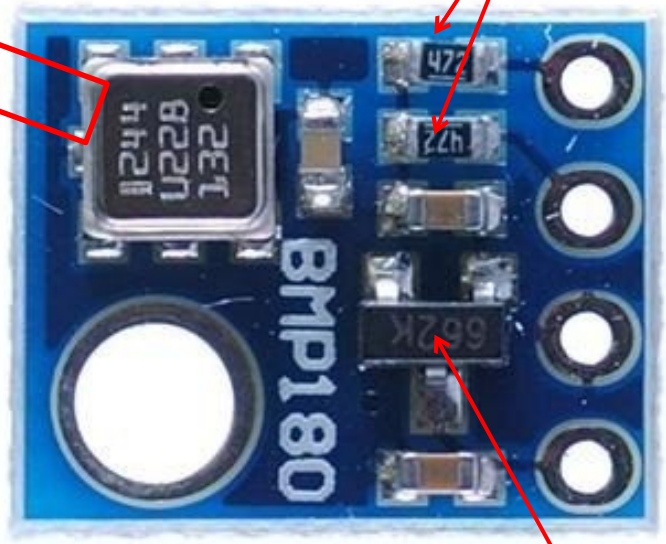


BMP180: nyomás és hőmérséklet mérés



Piezorezisztív nyúlásmérő, amely a nyomás hatására bekövetkező deformációt érzékeli

Felhúzó ellenállások



SDA

SCL

GND

VIN (+5V)

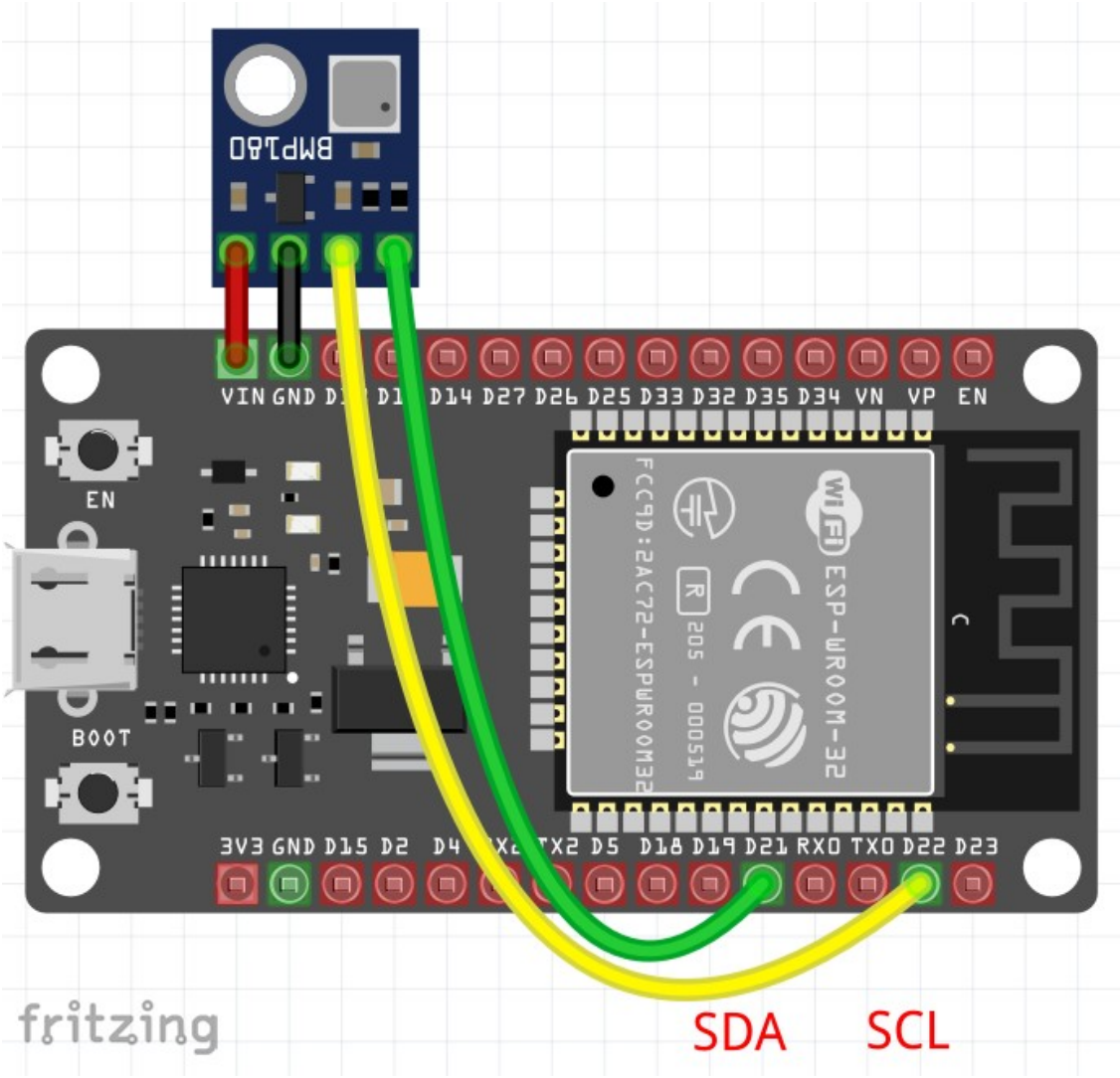
Feszültségstabilizátor (3,3V)

Bosch SensorTec BMP180

Nyomásmérés: 300 – 1100 hPa (9000 - -300m)
Tápfeszültség: 1,8 – 3,6 V
Áramfelvétel: 5 µA (1 mintavétel/s esetén)
Kis zaj: 0.06hPa (0.5m) kisfogyasztású mód
0.02hPa (0.17m) nagyfelbontású mód
Jellemzők: Hőmérő, I2C felület, gyárilag kalibrált

BMP180 bekötési vázlat

- A **BMP180** modul saját feszültségstabilizátort tartalmaz, ezért az 5 V-os tápfeszültségre kell kötni (ESP32 kártya Vin pontja)
- Az **ESP32** I2C kivezetései:
  SDA – GPIO21
  SCL – GPIO22



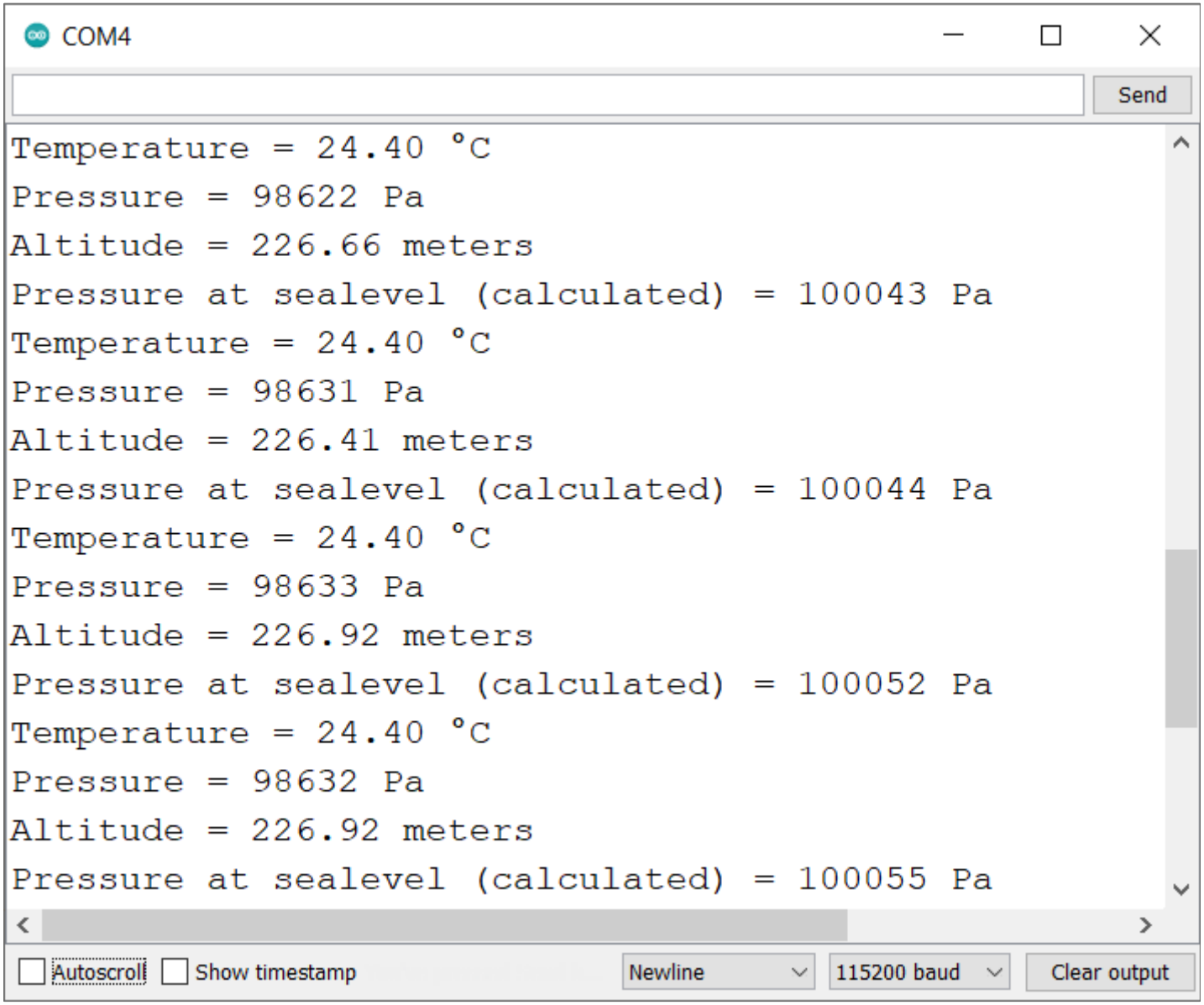
ESP32_BMP180.ino

```
#include <Wire.h>
#include <Adafruit_BMP085.h>
Adafruit_BMP085 bmp;

void setup() {
  Serial.begin(115200);
  if (!bmp.begin()) {
    Serial.println("Could not find a valid BMP085/BMP180 sensor, check wiring!");
    while (1) {}
  }
}

void loop() {
  Serial.print("Temperature = ");
  Serial.print(bmp.readTemperature());
  Serial.println(" °C");
  Serial.print("Pressure = ");
  Serial.print(bmp.readPressure());
  Serial.println(" Pa");
  Serial.print("Altitude = ");
  Serial.print(bmp.readAltitude());            // Assuming standard 101325 Pascal pressure
  Serial.println(" meters");
  Serial.print("Pressure at sealevel (calculated) = ");
  Serial.print(bmp.readSealevelPressure(120)); // 120 m above sea level (Debrecen)
  Serial.println(" Pa");
  delay(5000);
}
```

Szükségünk lesz az **Adafruit_BMP085** könyvtárra
Abban az alapértelmezett I2C cím 0x77

Ez így az egyezményes nyomásmagasság

ESP32_BMP180.ino futási eredmény



Helyi légnyomás átszámítása tengerszintre

Tudnunk kell a helyi tengerszint feletti magasságot (**altitude**) és a szenzorral meg kell mérnünk az abszolút helyi nyomás értékét (**p**).

A tengerszintre átszámított légnyomás ($\mathbf{p_0}$) a következő képlettel számítható ki:

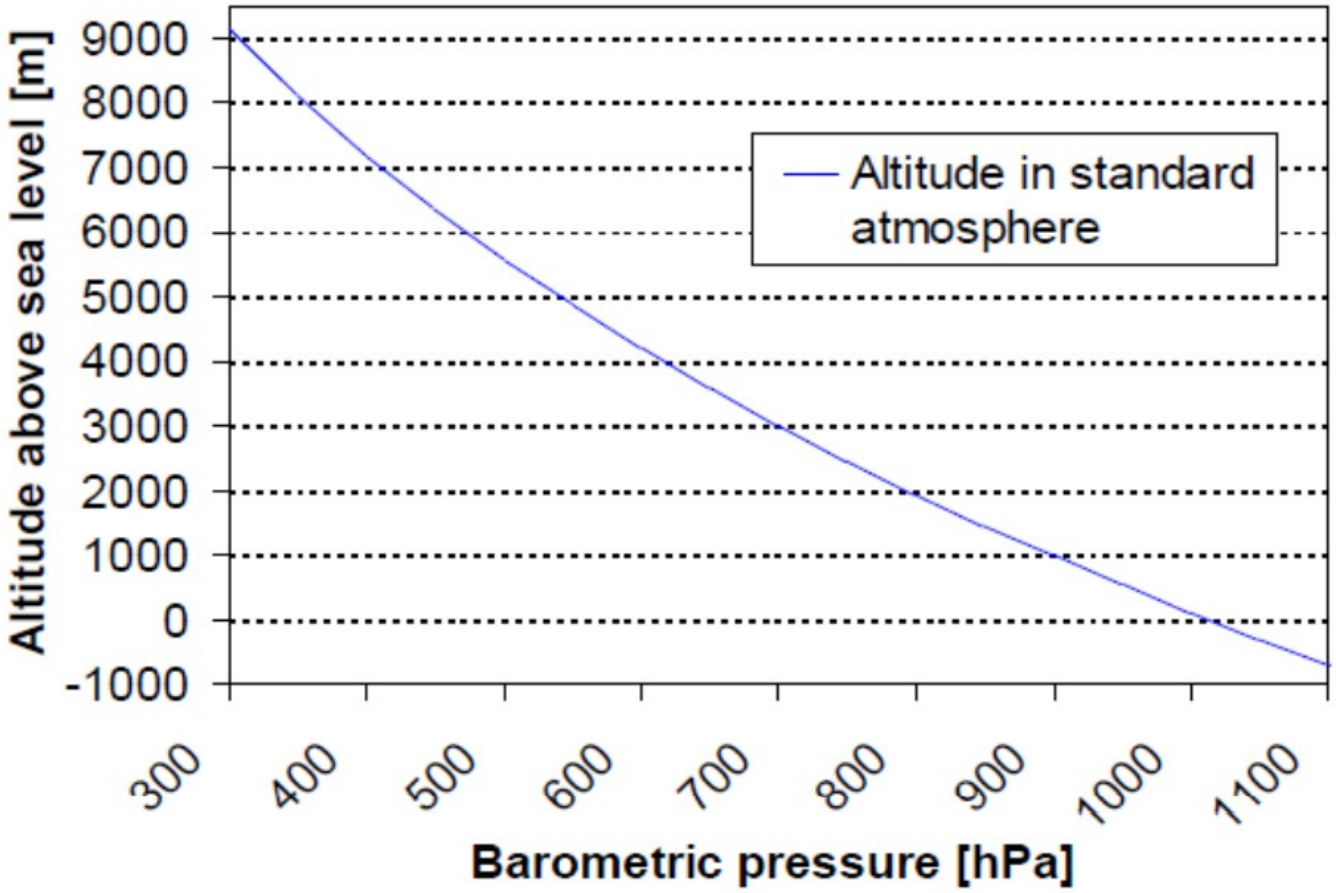
$$p_0 = \frac{p}{\left(1 - \frac{\text{altitude}}{44330}\right)^{5.255}}$$

Egyszerű(bb) közelítés:
Debrecenben (kb. 120 m) 1440 Pa-t hozzáadunk a mért légnyomáshoz

Magasság meghatározása

$$\text{magasság} = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

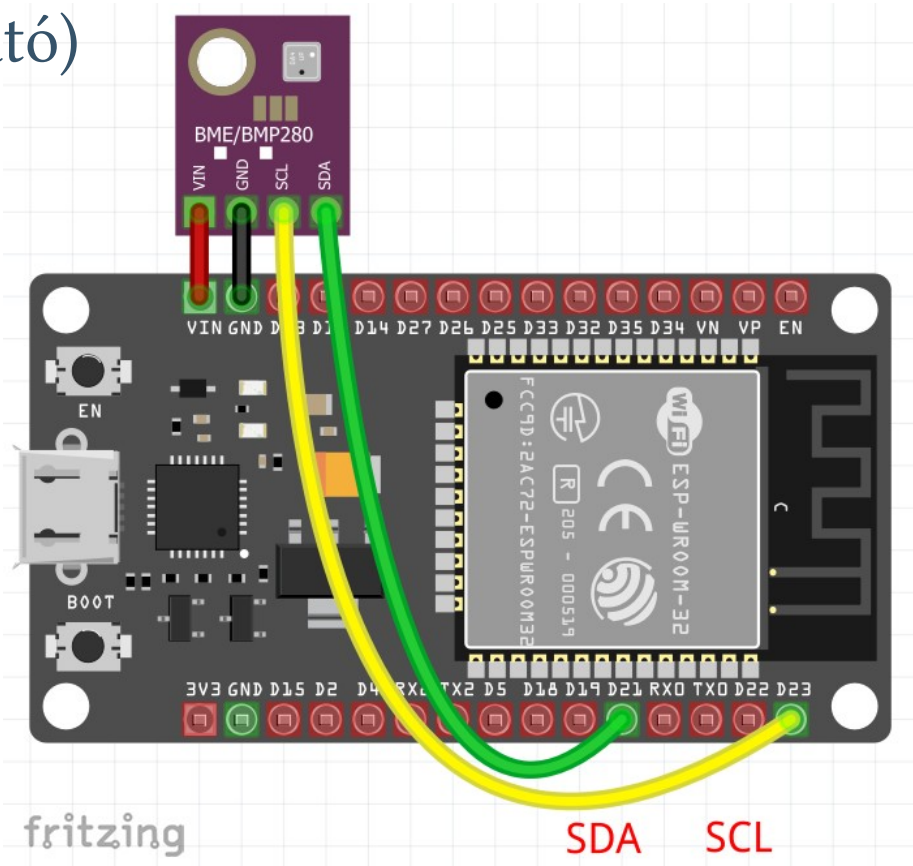
Ahol **p** az általunk mért nyomás, $\mathbf{p_0}$ pedig a tengerszinti nyomás (pl. 1013.25 hPa)



A gyakorlatban a nyomás nemcsak a magasságtól, hanem a meteorológiai viszonyoktól is függ (hőmérséklet, páratartalom, stb.)

A BME280 szenzor

- A BME típusjelzésben az E betű „environmental”, azaz környezeti szenzort jelez, ami a hőmérsékleten és légnyomáson kívül a levegő relatív páratartalmának mérésére is alkalmas
- Az általam használt szenzor modulon az **I2C cím 0x76,** tehát eltér az **Adafruit_BME280** programkönyvtár alapértelmezett címétől (A cím forrasztással megváltoztatható)
- A bekötés megegyezik a BMP180 szenzoréval
- A szenzor SPI módban is használható, erre itt most nem térünk ki



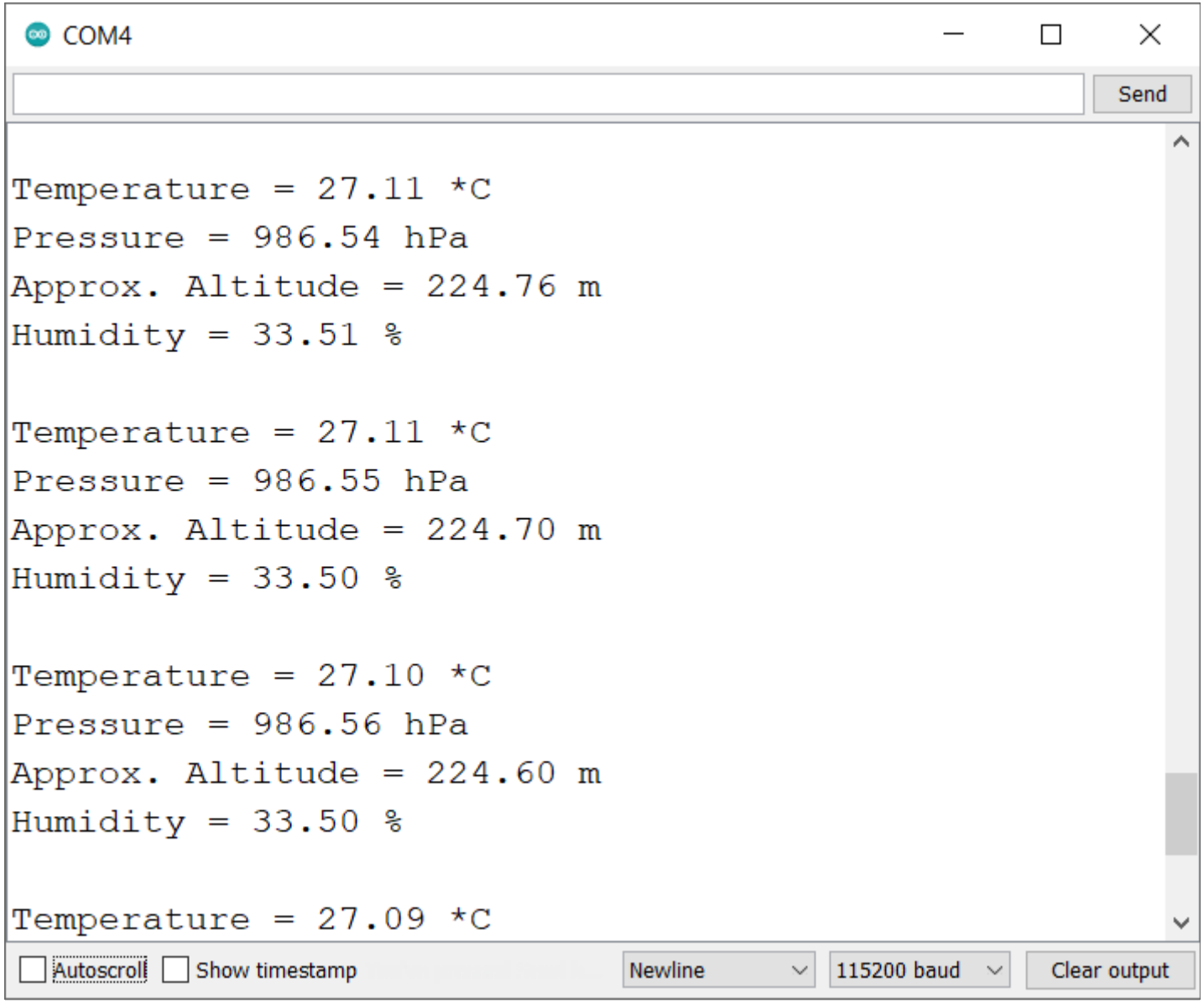
ESP32_BME280.ino

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme;           // I2C interface
unsigned long delayTime;

void setup() {
  Serial.begin(115200);
  while(!Serial);    // time to get serial running
  Serial.println(F("BME280 test"));
  unsigned status = bme.begin(0x76, &Wire);
  if (!status) {
    Serial.println("Couldn't find BME280 sensor, check wiring, address, sensor ID!");
        Serial.print("SensorID was: 0x"); Serial.println(bme.sensorID(),16);
        Serial.print(" ID of 0xFF probably means bad address, a BMP 180 or BMP 085\n");
        Serial.print(" ID of 0x56-0x58 represents a BMP 280,\n");
        Serial.print(" ID of 0x60 represents a BME 280.\n");
        Serial.print(" ID of 0x61 represents a BME 680.\n");
        while (1) delay(10);
    }
    Serial.println("-- Default Test --");
    delayTime = 1000;
    Serial.println();
}
```

ESP32_BME280.ino

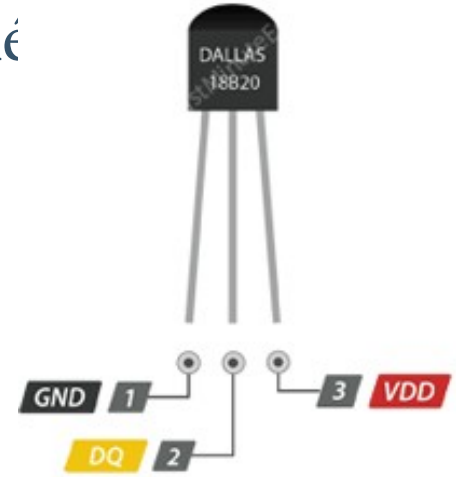
```
void loop() {
    printValues();
    delay(delayTime);
}


void printValues() {
    Serial.print("Temperature = ");
    Serial.print(bme.readTemperature());
    Serial.println(" *C");

    Serial.print("Pressure = ");

    Serial.print(bme.readPressure() / 100.0F);
    Serial.println(" hPa");

    Serial.print("Approx. Altitude = ");
    Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
    Serial.println(" m");

    Serial.print("Humidity = ");
    Serial.print(bme.readHumidity());
    Serial.println(" %");

    Serial.println();
}
```

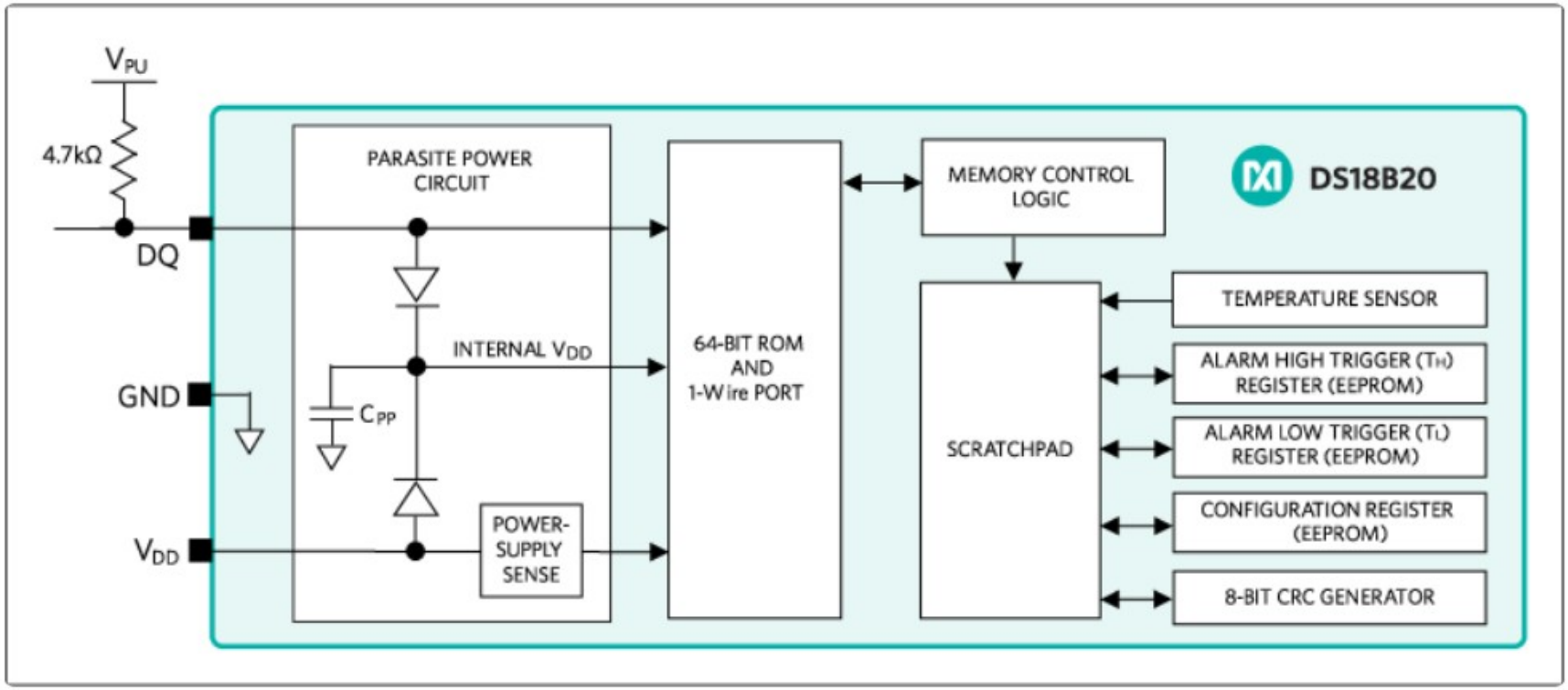
ESP32_BME280.ino futási eredmény



DS18B20

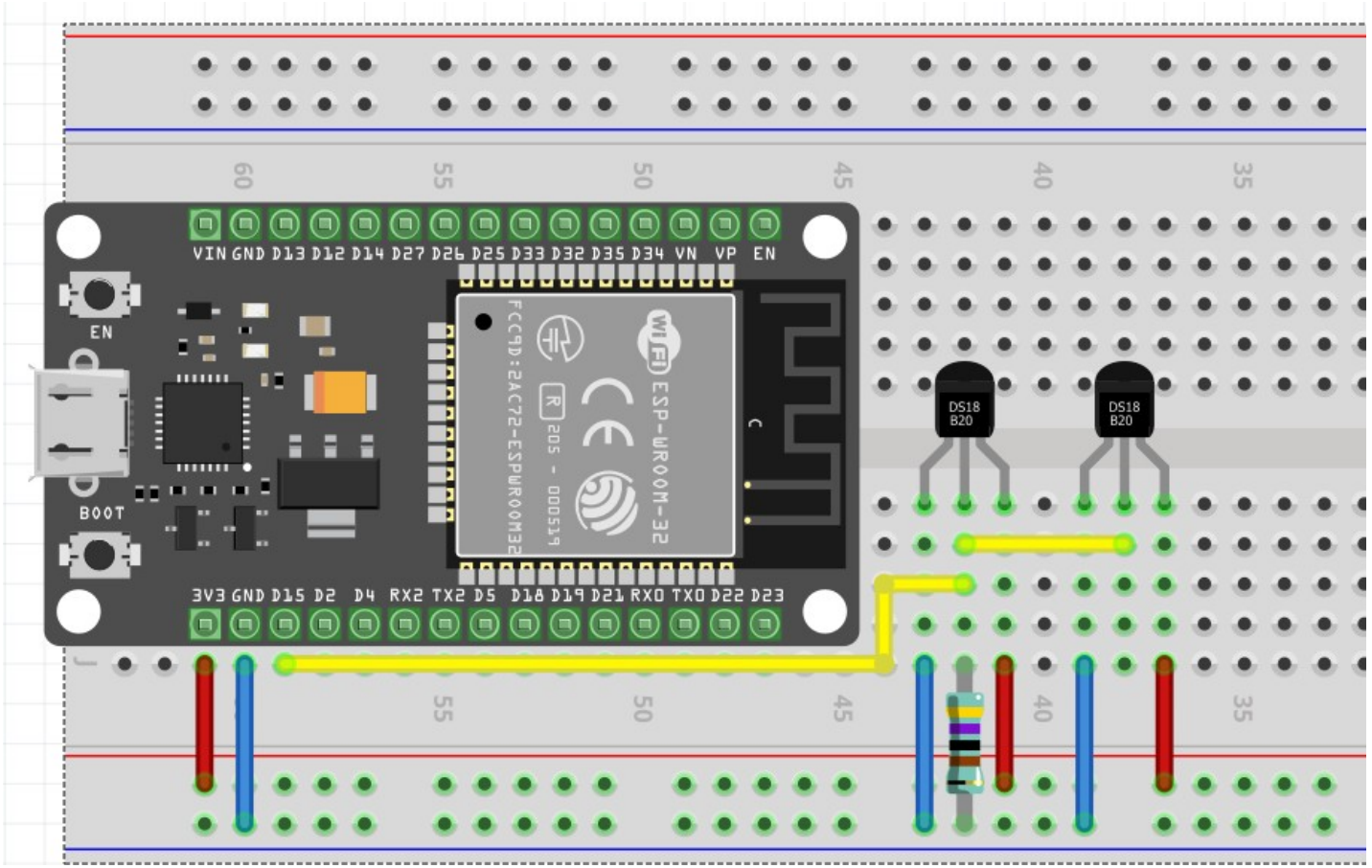
- Programozható felbontású (9 – 12 bit) digitális hőmé
- Kommunikáció: 1-wire protocol
- Mérési tartomány: –55°C – +125°C
- Pontosság: ±0.5°C (–10°C – +85°C tartományban)
- Tápfeszültség: +3.0V – +5.5V





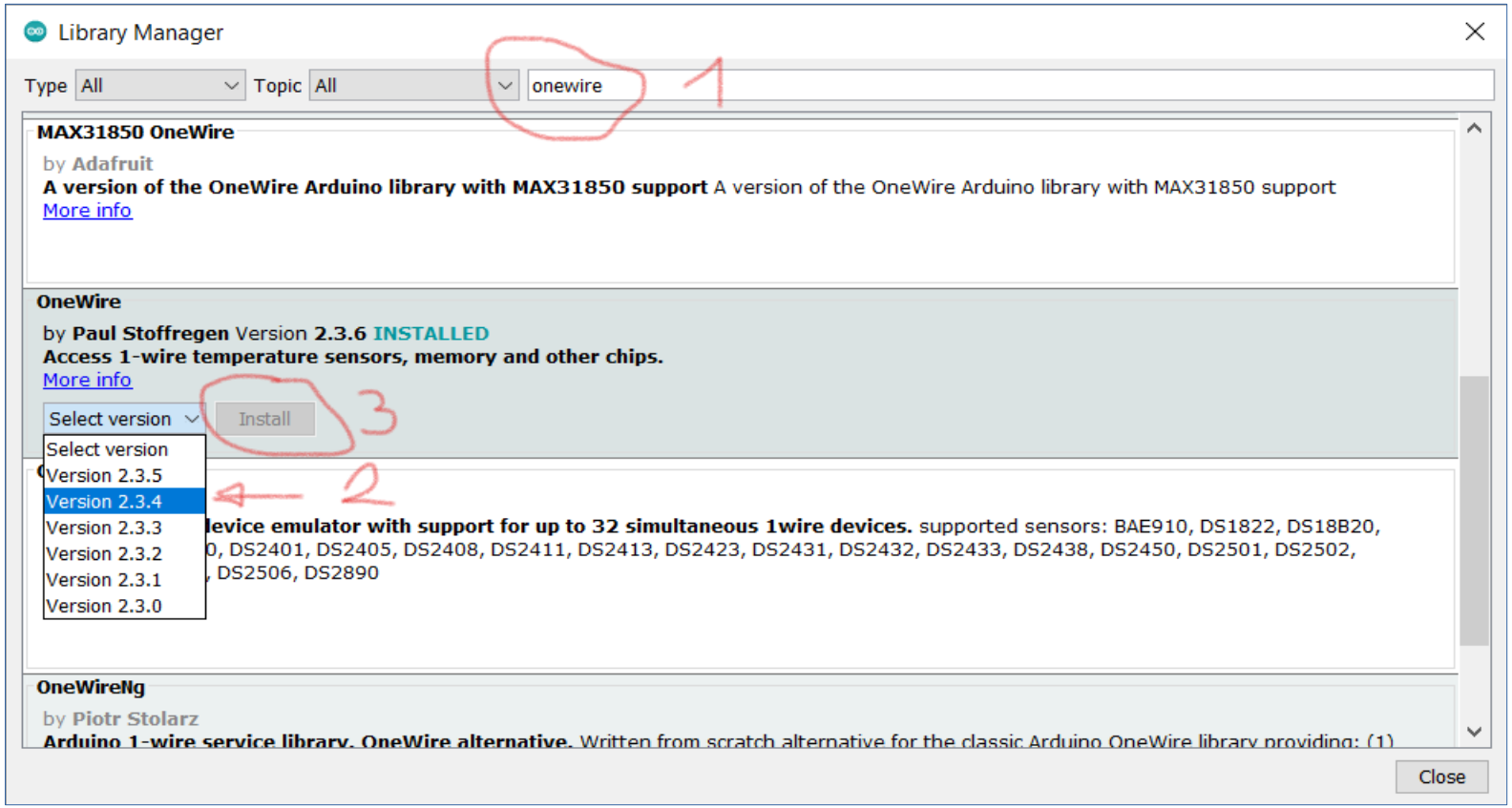
Kapcsolási vázlat

- Az **1-wire** buszra több eszköz is felfűzhető, s a sorrend, illetve a 64 bites azonosító alapján választhatjuk ki a kiolvasni kívánt eszközt
- Bármelyik kimenetre is alkalmas GPIO lábat használhatjuk, itt most a **GPIO15** kivezetést választottuk



Programkönyvtárak telepítése

- Nyissuk meg az Arduino IDE ***Tools → Manage Libraries*** menüjét!
- A kereső rovatba írjuk be az onewire keresőszót!
- Telepítsük az **OneWire** könyvtárat, illetve válasszuk ki a 2.3.4 (vagy egy korábbi) verziót és telepítsük újra!
- Telepítsük a **DallasTemperature** könyvtárat is



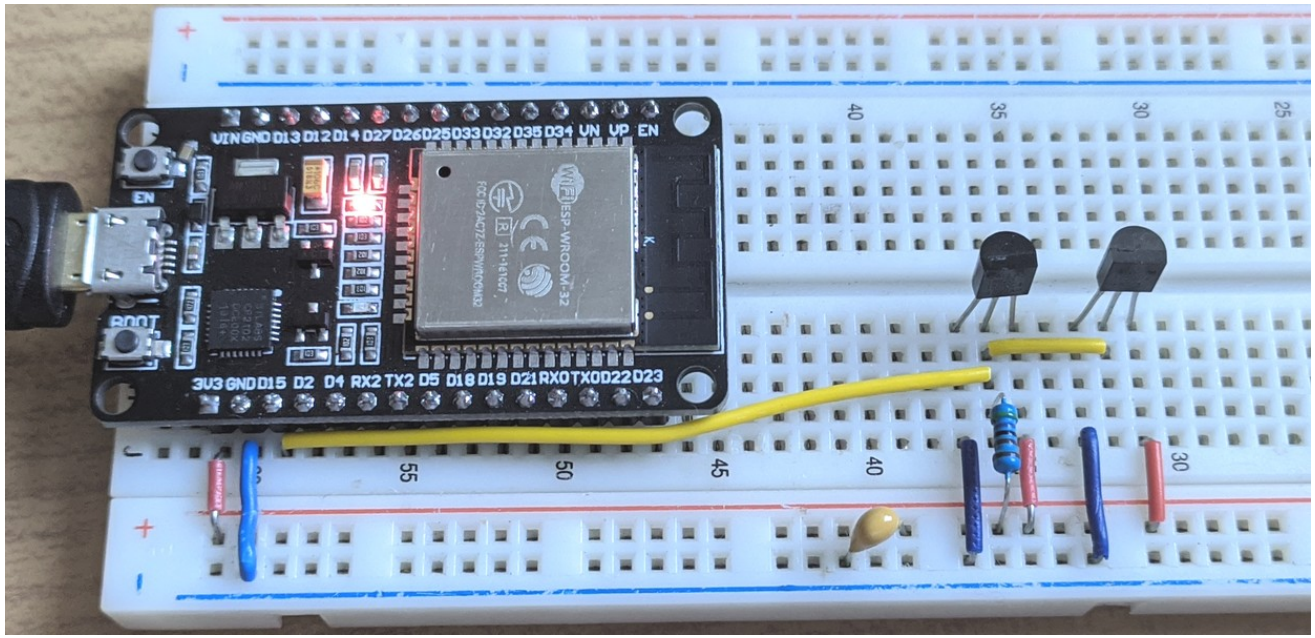
ESP32_DS18B20.ino

```
#include <OneWire.h>                    // OneWire 2.3.4 (Paul Stoffregen)
#include <DallasTemperature.h>          // DallasTemperature 3.8.0 (Miles Burton)

#define ONEWIREPIN 15                   // Az 1-wire busz a D15 lábra csatlakozik
int numberOfDevices;                    // Az 1-wire buszra kötött hőmérők száma

OneWire oneWire(ONEWIREPIN);            // 1-wire busz példányosítása
DallasTemperature sensors(&oneWire);    // hőmérő(k) példányosítása

void setup() {
  Serial.begin(115200);
  sensors.begin();                      // a szenzor(ok) inicializálása
  numberOfDevices = sensors.getDeviceCount();  // a szenzorok összeszámlálása
}

void loop() {
  sensors.requestTemperatures();        // Hőmérsékletmérés parancs kiküldése
  for(int i=0;i<numberOfDevices; i++){
    float tempC = sensors.getTempCByIndex(i);  // Az i. hőmérő kiolvasása
    Serial.print(i);                           // Az eredmény kiíratása
    Serial.print(". hőmérő: ");
    Serial.print(tempC,1);
    Serial.println("°C");
  }
  delay(5000);
}
```

Arduino IDE 1.8.16 (ezekkel működött)
ESP32 v2.0.0
OneWire 2.3.4 (vagy korábbi)
Dallas Temperature 3.8.0

ESP32_DS18B20 futási eredménye

- Kísérleteinkhez két DS18B20 hőmérő szenzort fűztünk fel az 1-wire buszra (többet is lehetne…)
- A kapcsolási elrendezés és a program futási eredménye az ábrákon látható

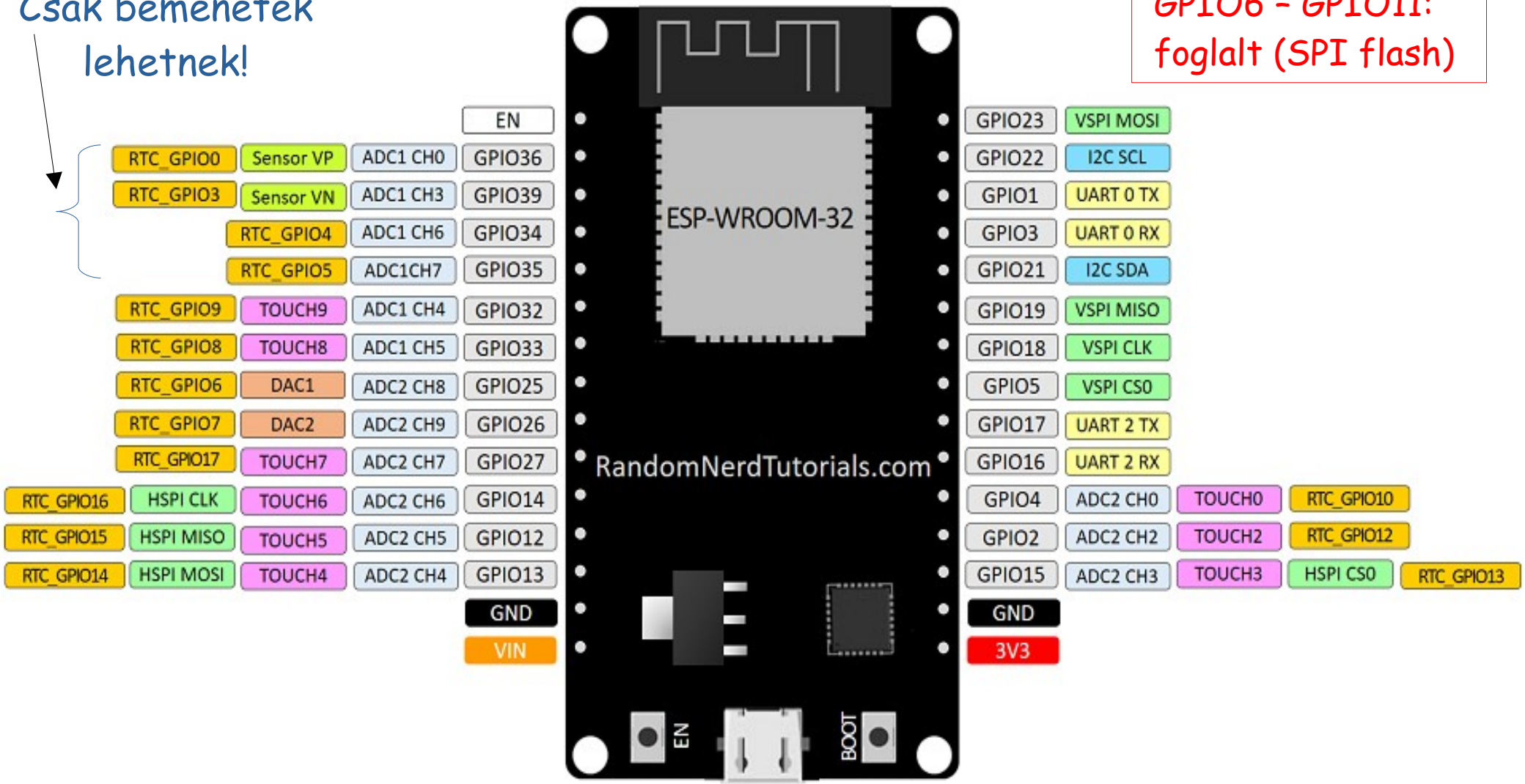



```
0. hőmérő: 27.5°C
1. hőmérő: 32.5°C
0. hőmérő: 27.5°C
1. hőmérő: 31.0°C
0. hőmérő: 27.5°C
1. hőmérő: 29.5°C
0. hőmérő: 27.5°C
1. hőmérő: 29.5°C
0. hőmérő: 27.5°C
1. hőmérő: 29.0°C
0. hőmérő: 27.5°C
1. hőmérő: 28.5°C
0. hőmérő: 27.5°C
1. hőmérő: 28.0°C
0. hőmérő: 27.5°C
1. hőmérő: 28.0°C
0. hőmérő: 27.5°C
1. hőmérő: 28.0°C
0. hőmérő: 27.5°C
1. hőmérő: 27.5°C
```

A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



- **Forrás:** randomnerdtutorials.com/getting-started-with-esp32/

Ellenállás színkódok